Procedure Development

Dr. Edward Nava
ejnava@unm.edu

Example code from Math.S

```
lw s0, numbers_to_use+0x00  #20
lw s1, numbers_to_use+0x04  #20
lw s2, numbers_to_use+0x08  #10
lw s3, numbers_to_use+0x0C  #10
lw s4, numbers_to_use+0x10  #5
lw s5, numbers_to_use+0x14  #5
lw s6, numbers_to_use+0x18  #0
lw s7, numbers_to_use+0x1C  #0
```

Text Segment Memory Contents

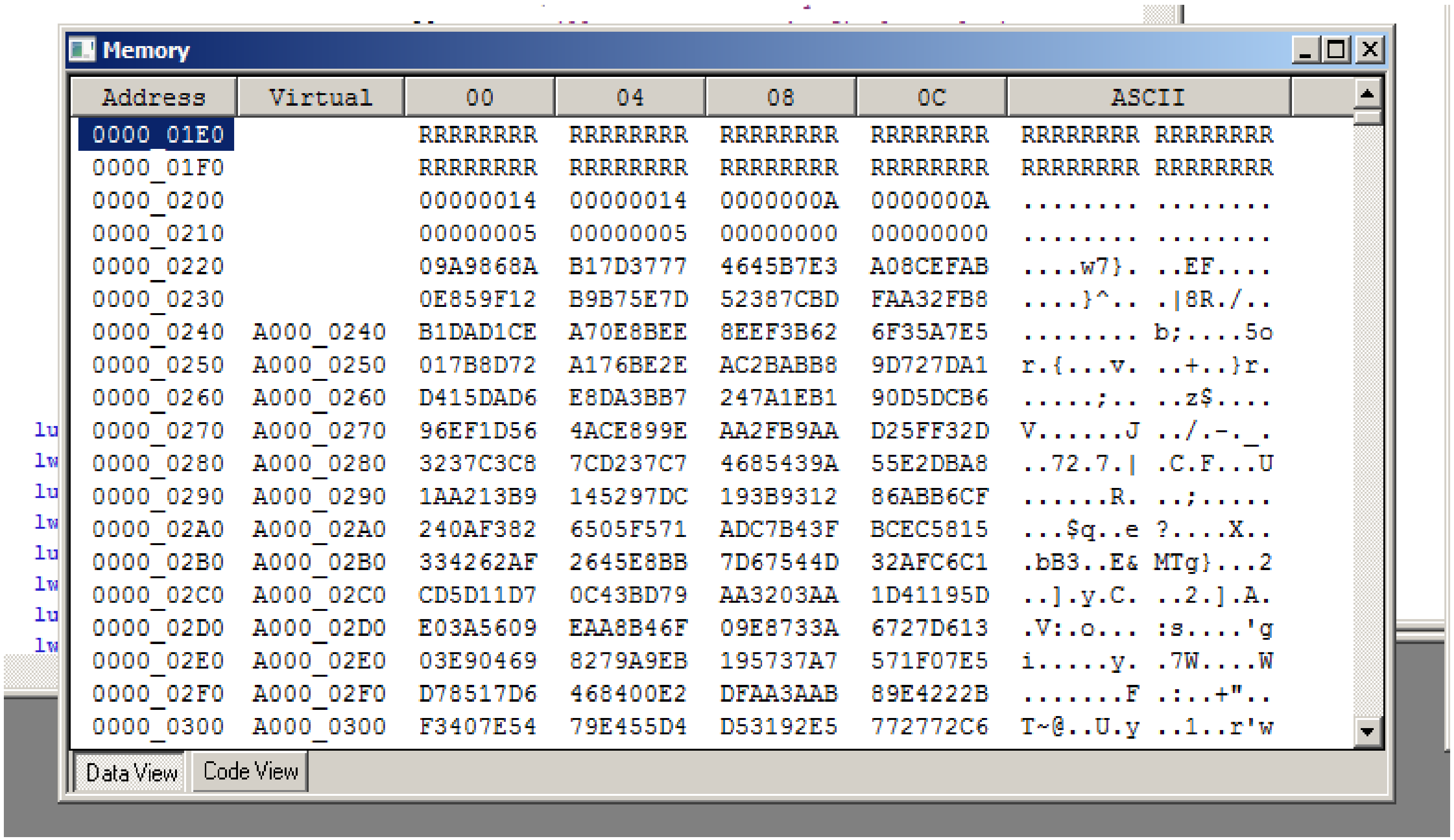
Virtual Address	Machine code	Instruction	
9D0000D8	3C10A000	lui	s0,0xa000
9D0000DC	8E100200	lw	s0,512(s0)
9D0000E0	3C11A000	lui	s1,0xa000
9D0000E4	8E310204	lw	s1,516(s1)
9D0000E8	3C12A000	lui	s2,0xa000
9D0000EC	8E520208	lw	s2,520(s2)
9D0000F0	3C13A000	lui	s3,0xa000
9D0000F4	8E73020C	lw	s3,524(s3)
9D0000F8	3C14A000	lui	s4,0xa000
9D0000FC	8E940210	lw	s4,528(s4)
9D000100	3C15A000	lui	s5,0xa000
9D000104	8EB50214	lw	s5,532(s5)
9D000108	3C16A000	lui	s6,0xa000
9D00010C	8ED60218	lw	s6,536(s6)
9D000110	3C17A000	lui	s7,0xa000
9D000114	8EF7021C	lw	s7,540(s7)

Code Segment Address Details

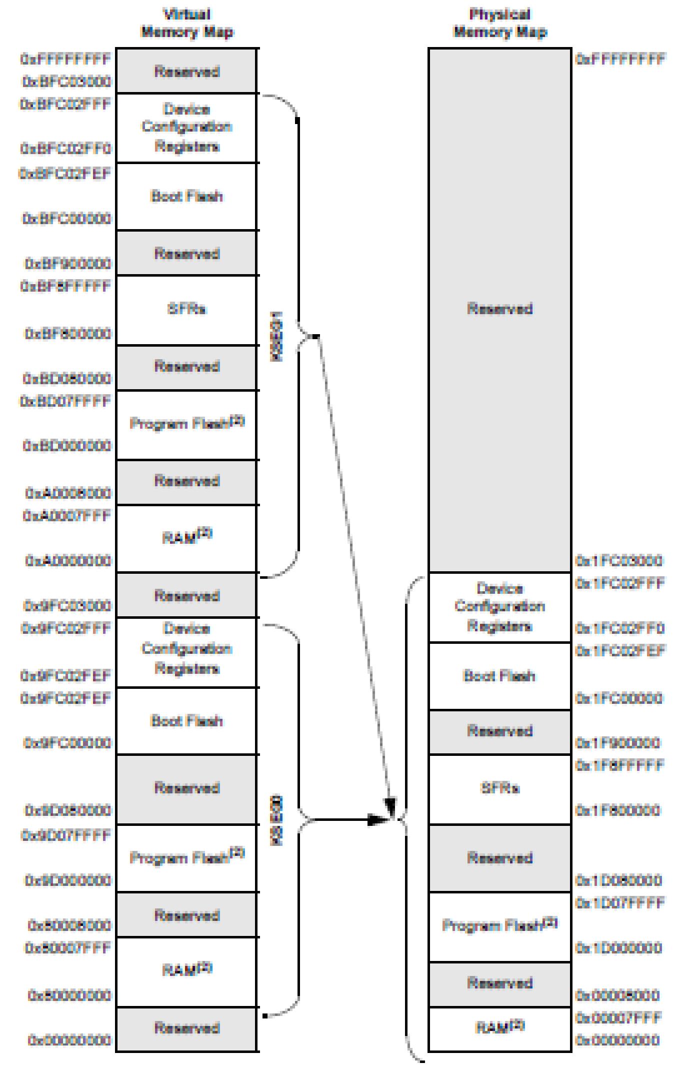
Memory

	Line	Address	Virtual	Opcode	Label	Disassembly
B	8249	1D00_00D8	9D00_00D8	3C10A000		lui s0,0xa000
	8250	1D00_00DC	9D00_00DC	8E100200		lw s0,512(s0)
	8251	1D00_00E0	9D00_00E0	3C11A000		lui s1,0xa000
	8252	1D00_00E4	9D00_00E4	8E310204		lw s1,516(s1)
B	8253	1D00_00E8	9D00_00E8	3C12A000		lui s2,0xa000
	8254	1D00_00EC	9D00_00EC	8E520208		lw s2,520(s2)
	8255	1D00_00F0	9D00_00F0	3C13A000		lui s3,0xa000
	8256	1D00_00F4	9D00_00F4	8E73020C		lw s3,524(s3)
	8257	1D00_00F8	9D00_00F8	3C14A000		lui s4,0xa000
	8258	1D00_00FC	9D00_00FC	8E940210		lw s4,528(s4)
	8259	1D00_0100	9D00_0100	3C15A000		lui s5,0xa000
	8260	1D00_0104	9D00_0104	8EB50214		lw s5,532(s5)
	8261	1D00_0108	9D00_0108	3C16A000		lui s6,0xa000
	8262	1D00_010C	9D00_010C	8ED60218		lw s6,536(s6)
	8263	1D00_0110	9D00_0110	3C17A000		lui s7,0xa000
	8264	1D00_0114	9D00_0114	8EF7021C		lw s7,540(s7)
	8265	1D00_0118	9D00_0118	24080000		addiu t0,zero,0
	8266	1D00_011C	9D00_011C	24090100		addiu t1,zero,256

Analyzing the Data Segment



PIC32MX460F512L Memory Map



Note 1: Memory areas are not shown to scale.

2: The size of this memory region is programmable (see **Section 3. "Memory Organization"** (DS61115)) and can be changed by initialization code provided by end-user development tools (refer to the specific development tool documentation for information).

Machine Language Instructions

- **When using an IDE or a simulator to examine memory contents, you will have to determine the corresponding assembly language instructions.**

- **We have three basic instruction formats (Register, Immediate, and Jump) which makes the decoding process fairly easy.**

Translating Assembly Language to Machine language - R-Type Instruction

Use the information in Appendix C of "MIPS Assembly Language Programming" to verify that 0x01024020
is the correct machine language encoding of the instruction
add \$8, \$8, \$2 add \$t0, \$t0, \$v0
In Appendix C we are shown how this instruction is encoded in binary
add Rd, Rs, Rt # RF[Rd] = RF[Rs] + RF[Rt]

Op-Code	Rs	Rt	Rd	Function Code
000000	sssss	ttttt	ddddd	00000100000

0000	0001	0000	0010	0100	0000	0010	0000
0x 0	1	0	2	4	0	2	0

Translating Assembly Language to Machine language

Use the information in Appendix C to verify that 0x3402000A is the correct machine language encoding of the instruction

ori $2, $0, 10

In Appendix C we are shown how this instruction is encoded in binary

ori Rt, Rs, Imm # RF[Rt] = RF[Rs] OR Imm

Op-Code	Rs	Rt	Imm
001101	sssss	ttttt	iiiiiiiiiiiiiiii

0011	0100	0000	0010	0000	0000	0000	1010
0x 3	4	0	2	0	0	0	A

Performance Evaluation

Figure of Merit (FM) measures commonly used are**:**

- Total clock cycles required to execute the code.
- Total number of memory locations required to store the code.

Assume:

- *Multiplication requires 32 clock cycles*
- *Division requires 38 clock cycles*

Functional Descriptions of Code Modules

- **A functional description will provide the information anyone needs to know if they are searching for a function that will be useful in solving some larger programming assignment.**
- **The functional description <u>only</u> describes what the function does, <u>not</u> how it is done.**
- **The functional description must explain how arguments are passed to the function and how results are returned.**
- **The following is an example functional description:**

Hexout($a0: value)

A 32-bit binary value is passed to the function in register $a0 and the hexadecimal equivalent is printed out right justified.

Implementing Functions

- **We can pass parameters to functions using registers $a0 - $a3**
- **We can also return parameter from functions using the registers $v0 and $v1.**
- **For proper operation, all developers must adhere to the register usage conventions.**
- **Usage conventions will require an ability to temporarily save register contents when using the $s0 - $s8 registers so that the content can be restored.**
- **The stack provides a means for passing data to functions and for temporarily storing register content.**

Passing Arguments on the Stack

An Example of Jack calling Jill(A, B, C, D, E)

```
addiu   $sp, $sp, -24   # Allocate Space on the Stack
sw      $t1,  0($sp)    # First In Parameter "A" at Mem[Sp]
sw      $t2,  4($sp)    # Second In Parameter "B" at Mem[Sp+ 4]
sw      $t3,  8($sp)    # Third In Parameter "C" at  Mem[Sp+ 8]
sw      $ra,  20($sp)   # Save Return address
jal     JILL            # Call the Function
lw      $ra, 20($sp)    # Restore Return Address to Main Program
lw      $t4, 12($sp)    # Get First Out Parameter "D"  at Mem[Sp+12]
lw      $t5, 16($sp)    # Get Second Out Parameter "E"  at Mem[Sp+16]
addiu   $sp, $sp, 24    # De-allocate Space on the Stack
```

Example of Jill accessing the Stack

JILL:

```
lw    $a0,  0($sp)    # Get First In Parameter "A" at Mem[Sp]
lw    $a1,  4($sp)    # Get Second In Parameter "B" at Mem[Sp+4
lw    $a2,  8($sp)    # Get Third In Parameter "C" at  Mem[Sp+8]
. . .
. . .       <Body of  Function>
. . .
sw    $v0, 12($sp)    # First Out Parameter "D"  at Mem[Sp+12]
sw    $v1, 16($sp)    # Second Out Parameter "E"  at Mem[Sp+16]
jr    $ra             # Return to JACK
```

Example of Jack Saving Important Temporary Registers

```
addiu   $sp, $sp, -32    # Allocate More Space on the Stack <####
sw      $t1,  0($sp)     # First In Parameter "A" at Mem[Sp]
sw      $t2,  4($sp)     # Second In Parameter "B" at Mem[Sp+ 4]
sw      $t3,  8($sp)     # Third In Parameter "C" at  Mem[Sp+ 8]
sw      $ra,  20($sp)    # Save Return address
sw      $t8,  24($sp)    # Save $t8 on the stack <####
sw      $t9,  28($sp)    # Save $t9 on the stack <####
jal     JILL             # call the Function
lw      $t8,  24($sp)    # Restore $t8 from the stack <####
lw      $t9,  28($sp)    # Restore $t9 from the stack <####
lw      $ra, 20($sp)     # Restore Return Address to Main Program
lw      $t4, 12($sp)     # Get First Out Parameter "D"  at Mem[Sp+12]
lw      $t5, 16($sp)     # Get Second Out Parameter "E"  at Mem[Sp+1
addiu   $sp, $sp, 32     # De-allocate Space on the Stack <####
```

Passing Arguments on the Stack